Multi-Task Reinforcement Learning with Soft Modularization

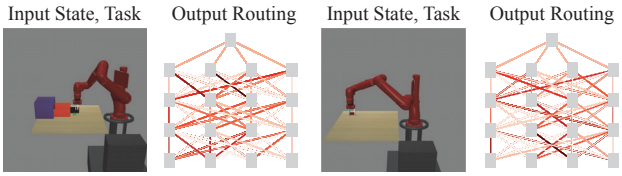
Ruihan Yang [1] **Huazhe Xu** [2] **Yi Wu** [3] **Xiaolong Wang** [1 2]

Abstract

Multi-task learning is a very challenging problem in reinforcement learning. While training multiple tasks jointly allow the policies to share parameters across different tasks, the optimization problem becomes non-trivial: It is unclear what parameters in the network should be reused across tasks, and the gradients from different tasks may interfere with each other. Thus, instead of naively sharing parameters across tasks, we introduce an explicit modularization technique on policy representation to alleviate this optimization issue. Given a base policy network, we design a routing network which estimates different routing strategies to reconfigure the base network for each task. Instead of creating a concrete route for each task, our task-specific policy is represented by a soft combination of all possible routes. We name this approach *soft modularization*. We experiment with multiple robotics manipulation tasks in simulation and show our method improves sample efficiency and performance over baselines by a large margin. Our project page is at: https://rchalyang.github.io/SoftModule.

1. Introduction

Deep Reinforcement Learning (RL) has recently demonstrated extraordinary capabilities in multiple domains, including playing games (Mnih et al., 2013), visual navigation (Zhu et al., 2017) and robotic control and manipulation (Lillicrap et al., 2015; Levine et al., 2016). Despite its successful applications, deep RL still requires a large amount of data for training, and the required sample size increases as the task become more complex. On the other hand, while the current deep RL methods can learn individual policies for specific tasks such as robot grasping and pushing, it remains very challenging to train a single network that generalizes across all possible robotic manipulation tasks. However, if we want our trained robot to actively interact with a complex environment (i.e., the real world), developing an algorithm that can learn a single model for multiple tasks simultaneously is what we must achieve.



(a) Open a draw (b) Pick and place

Figure 1: We design a multi-task policy network with soft modularization for robotics manipulation. Given different tasks (e.g., (a) open a draw and (b) pick and place), our network automatically estimate different routing paths to compute a soft combination of network modules. We use gray squares to represent the network modules and different color lines to represent the connection weights between modules (Darker the color indicates larger weight).

In this paper, we study multi-task reinforcement learning as one step forward towards skill sharing across diverse tasks and ultimately building robots that can generalize. By training deep networks with multiple tasks jointly, the network can learn to share and re-use components across different tasks, which further leads to improved sample efficiency. This is particularly important when we want to adopt RL algorithms in real-world applications. Multi-task learning also provides a natural curriculum since learning easier tasks can be beneficial to the learning of more challenging tasks with shared parameters. For example, Pinto & Gupta (2017) have shown that learning robot pushing and grasping together can improve the sample efficiency as well as the final success rate compared to training two tasks separately.

However, multi-task reinforcement learning remains a hard problem. It becomes even more challenging when the number of tasks increases. For instance, it has been shown by Yu et al. (2019) that training with diverse robot manipulation tasks jointly with a sharing network backbone and multiple task-specific heads for actions hurt the final performance comparing to independent training in each task. One major reason is that multi-task learning introduces optimization difficulties: It is unclear how the tasks will affect each other when trained jointly, and optimizing some tasks can bring

[1]UC San Diego [2]UC Berkeley [3]OpenAI. Correspondence to: Xiaolong Wang <xiw012@ucsd.edu>.

Preliminary work.

negative impacts on the others (Teh et al., 2017).

To tackle this problem, the compositional model with multiple modules was introduced (Andreas et al., 2017; Haarnoja et al., 2018a). For example, Andreas et al. (2017) proposed to train modular sub-policies and task-specific high-level policies jointly in a Hierarchical Reinforcement Learning (HRL) framework. The sub-policies can be shared across different high-level policies with a learned policy composition function. By enforcing this compositional structure, the trained networks achieve better performances than both independent task-specific policies and a full-shared multi-task policy in a maze environment. However, HRL introduces another optimization challenge on jointly training sub-policies and high-level task-specific policies together. Specifically, training sub-policies often require predefined tasks or some sophisticated way to discover subgoals for policy learning. Moreover, it may be challenging to even explicitly define subgoals in complex robotic manipulation tasks with a continuous state space.

Instead of designing individual modules explicitly for each sub-policy, we propose a soft modularization method in this paper, where we generate soft combinations of different modules for different tasks without explicitly specifying the policy structure. By using soft modularization, our approach learns to share modules across tasks automatically. Our approach consists of two networks: a base policy network and a routing network. The base policy network is composed of multiple modules. It takes the state as the input and outputs the action depending on the task. The routing network takes a task embedding together with the current state as input and estimates the routing strategy. When a new task is given, the modules in the base policy network will be reconfigured by the routing network. Figure 1 visualizes different connection configurations for different tasks. Furthermore, instead of taking hard assignments on modules for each task in the base network, our routing network outputs a probability distribution over module assignments for each task. A task-specific policy network can be viewed as a weighted combination of the shared modules according to the probability distribution. In this way, we can directly back-prop through the routing weights and train the routing network and base policy network together over multiple tasks. The advantage of our approach is that we can still modularize the networks according to different tasks without the need to specify policy hierarchies explicitly (e.g., HRL), which typically leads to optimization difficulties.

We perform experiments in the Meta-World environment (Yu et al., 2019), which contains 50 different robotic manipulation tasks. By using soft modularization, we achieve significant improvements in both sample efficiency and final performance over previous state-of-the-art multi-task policies. Although our approach utilizes far less training data compared to training individual policies for each task, our learned policy is still able to perform reasonably close to the individually trained policies. This shows that enforcing compositionality with soft modularization can improve the generalization across different tasks in RL.

2. Related Work

Multi-task learning. Multi-task learning (Caruana, 1997) is an important problem in machine learning. Researchers have shown how learning with multiple objectives can make different task benefit from each other in the field of computer vision (Misra et al., 2016; Zamir et al., 2018; Kokkinos, 2017), robotics and reinforcement learning (Wilson et al., 2007; Pinto et al., 2016; Pinto & Gupta, 2017; Riedmiller et al., 2018; Hausman et al., 2018; Sax et al., 2019). While sharing common parameters and structure across tasks can intuitively improve data efficiency, gradients from different tasks can cause negative interference on each other. This phenomenon is more severe in the context of RL. One way to avoid this negative interference is to use policy distillation (Parisotto et al., 2015; Rusu et al., 2015; Teh et al., 2017). For example, Teh et al. (2017) propose to share a distilled policy capturing common properties across tasks, and then train different task-specific policies by constraining them close to the shared policy. However, this line of approaches still requires separate networks for different policies and an extra distillation stage in learning.

Study on multi-task gradients. To handle the optimization problem in multi-task learning, researchers propose to explicitly model the similarity between gradients from different tasks (Zhang & Yeung, 2014; Chen et al., 2017; Kendall et al., 2018; Lin et al., 2019; Sener & Koltun, 2018; Du et al., 2018; Yu et al., 2020; Hu et al., 2019). For example, Chen et al. (2017) propose to normalize the gradients from different tasks for balancing among the multi-task losses. Besides balancing different tasks, Du et al. (2018) propose to compute the cosine similarity between gradients from the auxiliary task and the main task and use the similarity to adapt auxiliary tasks. This idea is then extended by Lin et al. (2019) to improve the sample efficiency in reinforcement learning. Besides adjusting the losses, Yu et al. (2020) propose to reduce the conflicting gradient directly after computing the gradient similarity. However, optimization relying on the gradient similarity is usually unstable, especially in the case where there is a large gradient variance within each task itself.

Compositional learning. Besides focusing on the gradients, another direction is to design compositional model with multiple modules (Singh, 1992; Devin et al., 2017; Andreas et al., 2017; Rusu et al., 2016; Qureshi et al., 2020; Peng et al., 2019; Haarnoja et al., 2018a; Sahni et al., 2017). Devin et al. (2017) proposes to decompose the network poli-

cies to task-specific and robot-specific modules for robotics manipulation tasks, and show that they can train and test on different combinations of modules. Motivated by this work, Andreas et al. (2017) further extends to train sub-policies for different subgoals and task-specific policies to combine sub-policies in a hierarchical reinforcement learning framework. Instead of training high- level task-specific policies, Peng et al. (2019) introduce to learn a gating function to output the mixture weights for composing the sub-policies. Despite these approaches show impressive generalizability over different tasks, they all required to pre-define and pre-train each individual sub-policy. This can be very challenging in the tasks where subgoals are not well defined.

Soft modularization. Instead of learning each individual sub-policy, our method factorizes the network into different modules without explicitly specifying their functions, and train a routing function to estimate a soft combination of the modules for each task. Our work is closely related to (Rosenbaum et al., 2017; 2019; Purushwalkam et al., 2019; Wang et al., 2019). Rosenbaum et al. (2019) propose to train routing policies with RL to select a model path to compose the modules. Instead of using RL to train the routing network, our soft modularization approach allows the routing network directly to be trained with back-propagation.

3. Background

We consider a finite horizon Markov decision processe (MDP) for each task $\mathcal{T}$ and there are M tasks in total, which can be represented by a tuple (S, A, P, R, H, γ), where the state $s \in S$ and action $a \in A$ are continuous. The transition probability $P(s_{t+1}|s_t, a_t)$ represents the stochastic transition dynamics. $R(s_t, a_t)$ represents the reward function. H is the horizon and γ is the discount factor. We use $\pi_\phi(a_t|s_t)$ to represent the policy whose network is parameterized by ϕ and the goal is to learn a policy maximizing the expected return. In the case of multi-task setting, we assume the tasks are sampled from a distribution $p(\mathcal{T})$, and different tasks will have different MDPs. In this paper, we train our RL policy with Soft Actor-Critic (SAC).

3.1. Reinforcement Learning with Soft Actor-Critic

Soft Actor-Critic (Haarnoja et al., 2018b) is an off-policy actor-critic deep reinforcement learning approach. In this framework, the actor aims to succeed at the task and act as randomly as possible. We consider the parameterized state value function as $V_\theta(s_t)$ and the soft Q-function is $Q_\theta(s_t, a_t)$. The parameters of these networks are θ. There are three types of parameters to optimize in SAC: The policy parameters ϕ, the parameters of Q-function θ and a temperature α. The objective of policy optimization is:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_\phi} [\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)] \right], \tag{1}$$

where α is a learnable temperature which is served as a entropy penalty coefficient. It can be automatically adjusted to maintain the entropy level of the policy. The optimization loss for the temperature α is:

$$J(\alpha) = \mathbb{E} a_t \sim \pi_\phi \left[-\alpha \log \pi_\phi(a_t|s_t) - \alpha \bar{\mathcal{H}} \right], \tag{2}$$

where $\bar{\mathcal{H}}$ is a desired minimum expected entropy. If $\log \pi_t(a_t|s_t)$ is optimized to increase its value, and the entropy is becoming smaller, α will be adjusted to increase in the process. The objective for learning Q-function is:

$$J_Q(\theta) = \tag{3}$$
$$\mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \left(R(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P} V_{\bar{\theta}}(s_{t+1}) \right) \right)^2 \right]$$

where the value function $V_{\bar{\theta}}$ is implicitly parameterized by the soft Q-fuction.

3.2. Multi-task Reinforcement Learning

We can easily extend SAC from single task to multi-task. The goal here is to learn a single, task-conditioned policy $\pi(a|s, z)$, where z represents an embedding of the task index. We optimize the policy to maximize the the average expected return across all tasks sampled from $p(\mathcal{T})$,

$$\mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} [\mathbb{E}_\pi [\sum_{t=0}^{H} \gamma^t R_t(s_t, a_t)]]. \tag{4}$$

The objective of policy optimization for multi-task learning can be represented as,

$$J_\pi(\phi) = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[J_{\pi, \mathcal{T}}(\phi) \right], \tag{5}$$

where $J_{\pi, \mathcal{T}}(\phi)$ is adopted directly from Eq. 1 with task $\mathcal{T}$. Similarly for Q-function, the objective is:

$$J_Q(\theta) = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})} \left[J_{Q, \mathcal{T}}(\theta) \right]. \tag{6}$$

4. Method

We propose to perform multi-task reinforcement learning by using a single base policy network with multiple modules. As visualized in Figure 2, instead of finding discrete routing paths to connect the modules for different tasks, we perform soft modularization: we utilize another routing network (on the right side of Figure 2) which takes the an task identity embedding and observed state as inputs, and outputs the probabilities to weight the modules in a soft manner.

With soft modularization, it allows task-specific policies to learn and discover what modules to share across different tasks. Since the soft combination process is differentiable, both policy network and the routing network can be trained together in an end-to-end manner. Note that the network architecture for the soft Q-function follows the similar structure, but initialized and trained with different parameters.

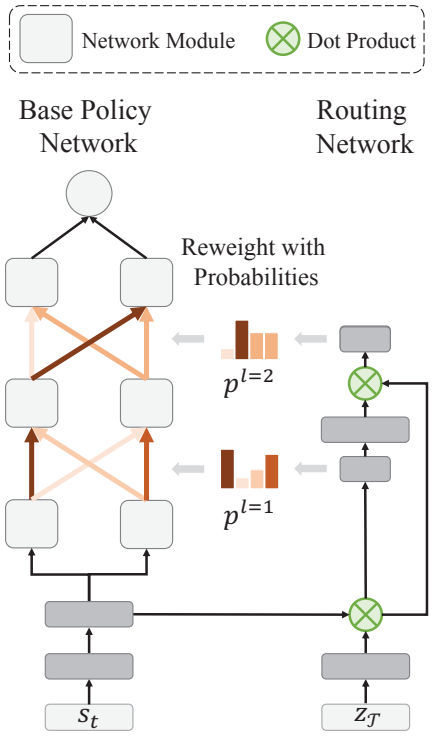


Figure 2: Our framework contains a base policy network with multiple modules (left) and a routing network (right). The routing network predicts $L-1$ layers of probabilities to weight the connections between different modules in the base policy network. The soft combinations of different modules are used to predict the action.

Although the soft modularization provides a differentiable way to modularize and share the network across tasks, different tasks can still learn and converge with different training speed based on the difficulties of the tasks. For example, learning "reaching" policy is usually much faster than learning "pick and place" policy. To tackle this problem, we introduce a simple way to automatically adjust the losses for different tasks to balance the training across tasks.

In the following subsections, we will first introduce our network architecture with soft modularization, and then training objective for multi-task learning with this architecture.

4.1. Soft Modularization

As shown in Figure 2, our model of multi-task policy contains two networks: the base policy network and the routing network. At each time stage, the network takes the input of the current state s_t and the task embedding $z_{\mathcal{T}}$ as inputs. We use an one-hot vector for $z_{\mathcal{T}}$ representing each task. We forward s_t to a 2-layer MLP and obtain a D-dimension representation $f(s_t)$, which is then used as inputs for the modules as well as the routing network. We extract the representation for the task embedding by one fully connected layer as $h(z_{\mathcal{T}})$, which is also in D-dimension.

Routing Network. The depth of our routing network is corresponding to number of module layers in the base policy network. Supposed we have L module layers and each layer has n modules in the base policy network. The routing network will have $L-1$ layers to output the probabilities to weight the modules and the dimension of the probability vector is $n \times n$. Formally, we define the output probability vector for the lth layer as $p^l \in \mathbb{R}^{n^2}$. The probability vector for the next layer can be represented as,

$$p^{l+1} = \mathcal{W}_d^l(\text{ReLU}(\mathcal{W}_u^l p^l \cdot (f(s_t) \cdot h(z_{\mathcal{T}})))), \quad (7)$$

where $\mathcal{W}_u^l$ is a fully connected layer in $\mathbb{R}^{D \times n^2}$ dimensions, which converts the probability vector to an embedding which has the same dimension as the task embedding representation and observation representation. We perform dot-product between these three embeddings to obtain a new feature representation. We adopt the dot-product here to combine the information from the previous probabilities, the task information and observation information. This feature is then forwarded to another fully connected layer $\mathcal{W}_d^l \in \mathbb{R}^{n^2 \times D}$, which leads to the probability vector for the next layer p^{l+1}. We visualize this process on computing $p^{l=2}$ from $p^{l=1}$ in Figure 2. For computing the first layer of probabilities, we use the inputs from both the task embedding and the state representation as,

$$p^{l=1} = \mathcal{W}_d^{l=1}(\text{ReLU}(f(s_t) \cdot h(z_{\mathcal{T}}))), \quad (8)$$

where $f(s_t)$ is the feature representation of the state with D dimensions. To weight the modules in the base policy network, we use softmax function to normalize p^l as,

$$\hat{p}_{i,j}^l = \frac{\exp(p_{i,j}^l)}{\sum_{j=1}^n \exp(p_{i,j}^l)}, \quad (9)$$

which is the probability of weighting the jth module in the lth layer for contributing to the ith module in the $l+1$ layer. We will illustrate how to use this probability in the base policy network in the following.

Base Policy Network. As shown in the left side of Figure 2, our base policy network has L layers of modules, and each layer contains n modules. We denote that the input for the jth module in the lth layer is a d-dimensional feature representation $g_j^l \in \mathbb{R}^d$. The input feature representation for the ith module in the $l+1$ layer can be represented as,

$$g_i^{l+1} = \sum_{j=1}^n \hat{p}_{i,j}^l(\text{ReLU}(W_j^l g_j^l)), \quad (10)$$

where $W_j^l \in \mathbb{R}^{d \times d}$ represents the parameters for the module . The features for the lower layer modules are first forwarded to different fully connected layers and a nonlinear layer (ReLU). We compute a weighted sum of these

features by using the probability outputs from the routing network. Recall from Eq. 9 that $\hat{p}^l_{i,j}$ represents the probability connecting the jth module in layer l to the ith module in layer $l+1$ and it is normalized to $\sum_j \hat{p}^l_{i,j} = 1$.

Given the features for the modules in the last layer, we compute the mean and variance as the outputs,

$$\mu, \sigma = \sum_{j=1}^{n} W_j^L g_j^L, \tag{11}$$

where $W_j^L \in \mathbb{R}^{d \times o}$ are the parameters for the modules in the last layer, and o represents the dimension of the outputs.

Note that although we have only introduced the network architectures for policies so far, we adopt similar architectures with soft modularization for Q-function as well. The weights for both the base policy network and the routing network are not shared or reused in the Q-function. During training, we train all the parameters jointly.

4.2. Multi-task Optimization

We focus on the problem of balancing the learning across different tasks, since different tasks are converging in different speed (e.g., easier tasks are usually converging faster). We propose to balance the learning by scaling the training objectives for the policy networks with different weights for different tasks. These weights can be learned automatically: the objective weight will be large if the policy for a task is not well trained and the objective weight will become small if the confidence of the policy is high.

This loss weight is directly related to the temperature parameter α in Soft Actor-Critic, which can be trained via Eq. 2. By optimizing Eq. 2, if the value of $\log \pi_\phi(a_t|s_t)$ is becoming larger, which means entropy is becoming smaller, α will become larger to encourage exploration. On the other hand, α will become small if $\log \pi_\phi(a_t|s_t)$ is small. In our multi-task setting, we have different temperature parameters for M different tasks: $\{\alpha_i\}_{i=1}^M$. The objective weights w_i for task i are proportional to the exponential of negative α_i,

$$w_i = \frac{\exp(-\alpha_i)}{\sum_{j=1}^{M} \exp(-\alpha_j)}. \tag{12}$$

We adjust the optimization objective from Eq. 5 as,

$$J_\pi(\phi) = \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\left[w_\mathcal{T} \cdot J_{\pi,\mathcal{T}}(\phi)\right], \tag{13}$$

and the objetive for Q-fuction from Eq. 6 is adjusted as,

$$J_Q(\theta) = \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\left[w_\mathcal{T} \cdot J_{Q,\mathcal{T}}(\theta)\right]. \tag{14}$$

5. Experiments

We perform experiments on multi-task robotics manipulation. We will first discuss the experimental environment, benchmark and baselines used, then we compare our method with the baselines and conduct ablation study.

5.1. Environment

We evaluate our approach with the recent proposed Meta-World (Yu et al., 2019) environment. This environment contains 50 different robotics continuous control and manipulation tasks with a sawyer arm in the MuJoCo environment (Todorov et al., 2012). There are two challenges for multi-task learning in this environment: i) the MT10 challenge, which requires learning 10 manipulation tasks simultaneously and ii) the MT50 challenge, which contains all the 50 manipulation tasks.

Building on top of these two challenges, we further extend the tasks to be goal-conditioned tasks. More specifically, the original MT10 and MT50 tasks are manipulation tasks with fixed goals. To make the tasks more realistic, we extend the tasks to have flexible goals. For example, in the case of reaching, instead of manipulating a robot arm to reach one fixed location, we extend the task to ask the robot to reach different goal location from the inputs. We name the two extensions as **MT10-Conditioned** and **MT50-Conditioned** tasks. To identify two variants of MT10 and MT50, we denote the original MT10 challenge as **MT10-Fixed** and the original MT50 challenge as **MT50-Fixed**.

5.2. Baselines and Experimental Settings

Baselines. We train our model with Soft Actor-Critic (SAC) (Haarnoja et al., 2018b). We compare to three baselines with SAC without using our network architecture as following:

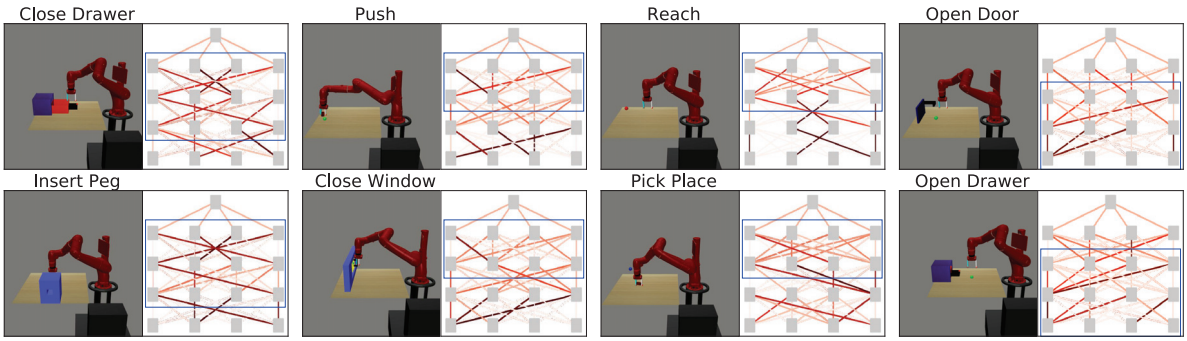
- **Single-task SAC**: we train individual policy with SAC for each task in the MT10-Conditioned setting.
- **Multi-task SAC (MT-SAC)**: using a one-hot task ID embedding together with the state as inputs.
- **Multi-task multi-head SAC (MT-MH-SAC)**: This baseline is built upon the previous Multi-task SAC baseline with one independent head for each task.

The same multi-task SAC and multi-task multi-head SAC baselines are also proposed in (Yu et al., 2019). We reproduce their results by following their settings.

Variants of Our Approach. We conduct all experiments under two settings of our method for better understanding of soft modularization. We ablate different numbers of module layer and modules in each layer:

- **Ours (Shallow)**: This model contains $L = 2$ module layers, $n = 2$ modules per layer and the output for each module is a $d = 256$ dimensions representation.
- **Ours (Deep)**: We propose a variant of the model with more module layers but maintaining same number of parameters. We have $L = 4$ module layers with $n = 4$ modules per layer. We set the input and output for each module a $d = 128$ dimensions representation.

Evaluation Metrics. We evaluate the policies based on the success rate of executing the tasks, which is well-defined in



(a) (b) (c) (d)

Figure 3: Sampled observation and corresponding routing visualization. Each column represents a pair of two different tasks sharing similar routing. We highlight the shared part with blue boxes. The pair of tasks include: (a) Close Drawer and Insert Peg; (b) Push and Close Window; (c) Reach and Pick Place. (d) Open Door and Open Drawer.

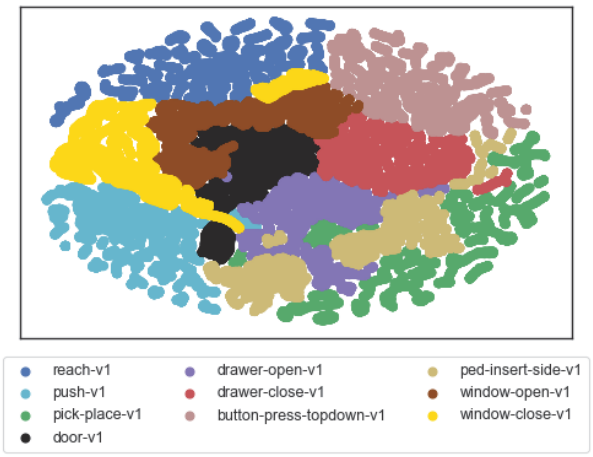


Figure 4: Routing visualization. We extract the probabilities predicted from the routing network for different tasks and visualize with t-NSE. We can see that the routing probabilities from different tasks are grouped in different clusters.

the Meta-World environment(Yu et al., 2019). We use the average success rate cross tasks to measure the performance. For each experiment, we train both our approaches and the baselines with 3 random seeds. To plot the training curves, we plot the success rate of the polices across time with variance. To report the final performance, we directly evaluate the final policy for each approach. We sample 100 episodes for each task and we repeat the same process for each seed. We compute the success rate for all these trials and report the averaged results.

Training samples. For fair comparison, all methods are evaluated after convergence. For baselines, we train them with 20 million samples on the MT10 setting and 100 million samples on the MT50 setting. For our method with soft modularization, it converges much faster than the baselines, and we train it with 15 million samples for MT10 and with 50 million samples for MT50 tasks.

5.3. Routing Network Visualization

We perform visualization on the networks trained with Ours (Deep) on the MT10-Conditioned setting.

Probability Visualization. We visualize the probabilities p^l predicted by the routing network. The Ours (Deep) model contains $l = 4$ module layers with $n = 4$ modules on each layer. As shown in Figure 3, we plot p^l as the connections between different modules and use deep red color to represent large probability and light red color to represent low probability. For each column, we visualize the routing networks for two different tasks. We can see that even the tasks are different, they can still share similar module connections. This shows our soft modularization method allows the reuse of skills across different manipulation tasks.

t-SNE Visualization. We visualize the routing probabilities for different tasks via t-SNE (van der Maaten & Hinton, 2008) in Figure 4. We run the policy on each task in MT10-Conditioned multiple times to collect routing data samples. We combine all the routing probabilities from all layers into a $(l-1)n^2 = 48$ dimensional vector representing the routing path and visualize via t-SNE in Figure 4. We find clear boundaries between tasks in routing space, indicating that the agent can distinguish different tasks and choose the corresponding skill set for each given task. Besides, we notice that those tasks sharing similar task structures (e.g., drawer-open-v1 and drawer-close-v1, window-open-v1 and window-close-v1) are close in the t-SNE plot.

5.4. Quantitative Results

Results on MT10-Fixed. As shown in Table 1c, our re-implementation of multi-task multi-head SAC performs very close to the reported results in (Yu et al., 2019). Although the final success rate of our method is only 2% better than our baseline, our method converges faster than the baseline, as shown in the 2nd plot in Figure 5. The reason that we

Method	MT50-Fixed	MT50-Conditioned
MT-SAC	28.8%	-
MT-SAC*	31.4%	28.3%
MT-MH-SAC	35.9%	-
MT-MH-SAC*	35.5%	34.2%
Ours (Shallow)	59.5%	60.4%
Ours (Deep)	**60.0%**	**61.0%**

(a) Average success rates over all tasks for MT50

Method	MT50-Fixed	Params	layers	units
MT-MH-SAC	35.9%	1.2x	3	400
MT-MH-SAC*	35.5%	1.2x	3	400
MT-MH-SAC-4*	46.7%	1.6x	4	400
MT-MH-SAC-5*	45.2%	2.0x	5	400
MT-MH-SAC-6*	45.0%	2.4x	6	400
MT-MH-SAC-4-Wide*	50.7%	3.3x	4	600
MT-MH-SAC-5-Wide*	50.3%	4.2x	5	600
Ours (Deep)	**60.0%**	1x	-	-

(b) Comparison with baselines using different number of parameters for MT50-Fixed.

Method	MT10-Fixed	MT10-Conditioned
MT-SAC	39.5%	-
MT-SAC*	44.0%	42.6%
MT-MH-SAC	**88.0%**	-
MT-MH-SAC*	85.0%	67.4
Ours (Shallow)	87.0%	**71.8%**
Ours (Deep)	86.7%	68.4%

(c) Average success rates over all tasks for MT10

Method	MT10-Conditioned
Single-task SAC	78.5%
Ours (Shallow)	71.8%

(d) Comparison on average success rate between the single task SAC policy and our multi-task policy for MT10-Conditioned.

Table 1: Comparisons on average success rates for MT10 and MT50 tasks. MT-SAC, MT-MH-SAC indicate results reported in (Yu et al., 2019). Approaches with * indicate baselines of our own implementation.

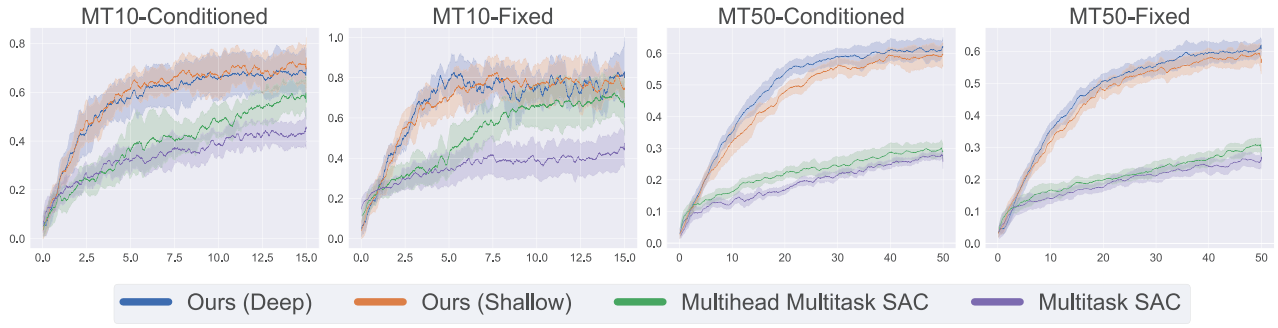


Figure 5: Training curves of four different methods on all benchmarks. The concrete lines represent the average over different seeds and the shaded areas represent the standard variance over seeds. For MT10, our method converges much faster than the baselines. For MT50, we achieve a large gain not only on sample efficiency but also on performance.

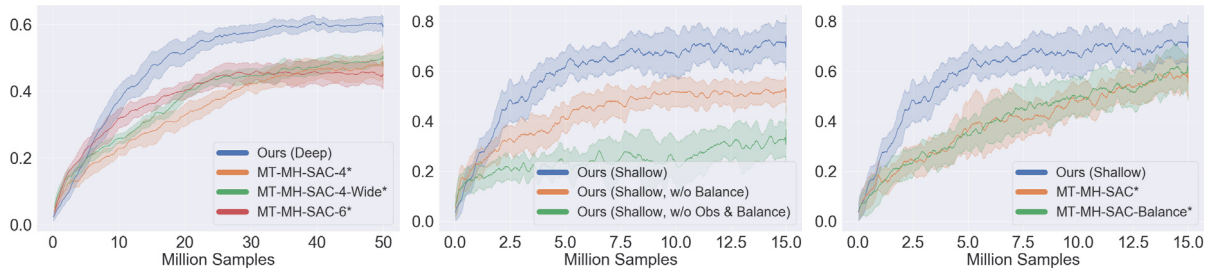
are not getting a significant gain in the final success rate is that training 10 tasks with fixed goals is quite simple. We move forward to a more practical and challenging setting with training goal-conditioned policies.

Results on MT10-Conditioned. As task difficulty increases, we can see from Table 1c that our approach (Ours (Shallow)) achieves more than 4% improvement over the baseline. Our approaches continue to improve the sample efficiency over the baselines (1st plot in Figure 5).

Results on MT50-Fixed and MT50-Conditioned. When we are moving from joint training with 10 tasks to 50 tasks, the problem becomes more challenging. As shown in Table 1a and the last two plots in Figure 5, our method achieves a significant improvement over the baseline methods (around 25%) in both the fixed goal and goal-conditioned settings. We also observe that in MT50 environments, Ours (Deep) performs better than Ours (Shallow) approach, while it is the opposite in the MT10 setting. The reason for this phenomenon might be: (i) for a smaller number of task (MT10), simple network topology facilitates more on information sharing across tasks; (ii) for larger number of task (MT50), more complex network topology provides more routing choices and prevents different tasks from harming the performance of each other. It is also worthy of mentioning that our method achieves better success rates on the MT50-Conditioned environments than MT50-Fixed. The reason is that MT50-Conditioned provides more examples in training for better generalization.

5.5. Effects on Network Capacity

We conduct experiments to see how the capacity of the network (number of parameters) can influence the performance of the baseline methods. We compare our approach with



(a) MT50-Conditioned (b) MT10-Conditioned (c) MT10-Conditioned

Figure 6: (a) Comparison on Ours (Deep) and baselines with different number of parameters for MT50-Conditioned. (b) Analysing the effects of balancing training examples and using observation for routing network in Ours (Shallow) for MT10-Conditioned. (c) Analysing the effect of balancing training examples in the baseline for MT10-Conditioned.

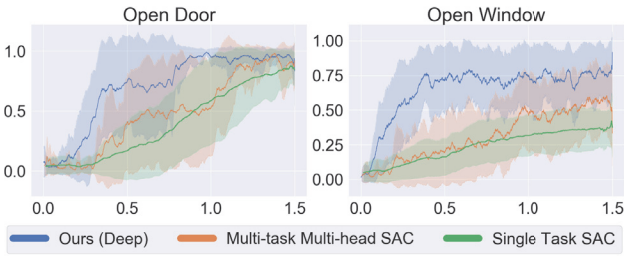


Figure 7: Comparison with Single Task SAC. X-axis denotes the number of samples from given tasks.

baselines using different numbers of parameters for MT50-Fixed in Table 1b. We ablate different number of network layers and the number of hidden units in each layer. We denote MT-MH-SAC-l^* as the multi-task multi-head SAC baseline with l layers. We also ablate more hidden unites for each layer and name the methods with "Wide". The detailed configurations for different ablations are shown in Table 1b.

We observe that even our model uses the smallest number of parameters, we can still achieve much better results. For example, our method is around 10% better than the baseline (MT-MH-SAC-5-Wide*) which has 4.2x number of parameters compared to our method. We also observe that the gain saturates very fast as we make the network larger and larger: The baseline with 4.2x capacity is slightly worse than the baseline with 3.3x capacity. We visualize the training curve in Figure 6a: our method converges faster and has much better performance than large capacity baselines.

5.6. Comparison with Single Task Policy

A substantial advantage of multi-task learning is with sample efficiency. We compare our policy with individual policy trained for each task. We select two tasks (open door and open window) from MT10-Conditioned and visualize the training curves in Figure 7. For our approach, we plot the performance obtained using the same amount of data for one specific task as the single task policy. It shows that for each task, using data from other tasks along with our method can significantly improve sample efficiency, and skills learned by the soft modular policy can be shared between tasks with routing. Of course, the single task policy can overfit easily given enough training examples and achieve a very good result for one specific task. In Table 1d, we show that our method can still perform reasonably close to the single task policy, even we train with much fewer examples with much fewer parameters via a shared network.

5.7. Analysing Learning Components

We analyse the importance of two learning components in our method with MT10-Conditioned: (i) Balance the training across different tasks by using the temperature parameters (Eq. 12); (ii) Use observation representation as the inputs for the routing network.

We report the comparison results in Figure 6b and Figure 6c. In Figure 6c, we ablate our method in the Ours (Shallow) setting, and remove the balance training (Ours (Shallow, w/o Balance)) as well as remove both the balance training and observation inputs for the routing network (Ours (Shallow, w/o Obs & Balance)). With our approach, we can reach around 70% success rate across 10 tasks. If we remove one or both learning components, the success rate is reduced by a large margin. Thus both components play an important role in our approach. We also add the balance training strategy in optimizing the baseline approach in Figure 6c as MT-MH-SAC-Balance. Interestingly, we find the baseline approach is not affected as much with the new optimization strategy. Thus we do not apply balance training for baselines.

6. Conclusion

In this paper, we propose multi-task reinforcement learning with soft modularization for robotics manipulation tasks. Our method improves the sample efficiency as well as the success rate over the baselines by a large margin. The advantage becomes more obvious when given more diverse tasks. This shows that soft modularization allows effective sharing and reusing network components across tasks, which opens up future opportunities to generalize the policy to unseen tasks in a zero-shot manner.

References

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017.

Caruana, R. Multitask learning. *Machine learning*, 28(1): 41–75, 1997.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176. IEEE, 2017.

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Pascanu, R., and Lakshminarayanan, B. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6244–6251. IEEE, 2018a.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018b. URL http://arxiv.org/abs/1801.01290.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. 2018.

Hu, H., Dey, D., Hebert, M., and Bagnell, J. A. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3812–3821, 2019.

Kendall, A., Gal, Y., and Cipolla, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6129–6138, 2017.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lin, X., Baweja, H., Kantor, G., and Held, D. Adaptive auxiliary task weighting for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4773–4784, 2019.

Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. Cross-stitch Networks for Multi-task Learning. In *CVPR*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Peng, X. B., Chang, M., Zhang, G., Abbeel, P., and Levine, S. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems*, pp. 3681–3692, 2019.

Pinto, L. and Gupta, A. Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2161–2168. IEEE, 2017.

Pinto, L., Gandhi, D., Han, Y., Park, Y.-L., and Gupta, A. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pp. 3–18. Springer, 2016.

Purushwalkam, S., Nickel, M., Gupta, A., and Ranzato, M. Task-driven modular networks for zero-shot compositional learning. *arXiv preprint arXiv:1905.05908*, 2019.

Qureshi, A. H., Johnson, J. J., Qin, Y., Henderson, T., Boots, B., and Yip, M. C. Composing task-agnostic policies with deep reinforcement learning. *ICLR*, 2020.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Rosenbaum, C., Cases, I., Riemer, M., and Klinger, T. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Sahni, H., Kumar, S., Tejani, F., and Isbell, C. Learning to compose skills. *arXiv preprint arXiv:1711.11289*, 2017.

Sax, A., Zhang, J. O., Emi, B., Zamir, A., Savarese, S., Guibas, L., and Malik, J. Learning to navigate using mid-level visual priors. *arXiv preprint arXiv:1912.11121*, 2019.

Sener, O. and Koltun, V. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, pp. 527–538, 2018.

Singh, S. P. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4): 323–339, 1992.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. 2008.

Wang, X., Yu, F., Wang, R., Darrell, T., and Gonzalez, J. E. Tafe-net: Task-aware feature embeddings for low shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1831–1840, 2019.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022, 2007.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. *arXiv*, 2020.

Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J., and Savarese, S. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

Zhang, Y. and Yeung, D.-Y. A regularization approach to learning task relationships in multitask learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3):1–31, 2014.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, pp. 3357–3364. IEEE, 2017.